Design Review 2

Apollinaire Mase (amm4zb)

Roman Boyarov (rb9ng)

Adam Abdulhamid (aaa7vc)

Overall, our group did great coming up with the design and looking at compromises and tradeoffs before deciding what to implement in our design. However, our inexperience at the software was our biggest crutch. Simulation proved to take much longer, if ever working fully the way we expected. Thus, our review is missing a good amount of simulation and that is our top priority to fix in the coming days. We are planning on spending a great amount of time familiarizing ourselves with Ocean simulation to make sure our circuits work as expected. We fully realize we need to get on top of this to ensure success on this project and have it on great concert get this done as soon as possible.

Towards the second design review, in terms of components for the ALU, we had to implement an adder and two's complement. Our mindsets towards the compromises we were going to make was to appropriately sacrifice area for speed. Before even simulating, we made the assumption that the adder was going to be the component that created the worst case delay for our ALU. Because we essentially use the adder as part of the process for the two's complement, that statement gets extended to that as well. This is all in comparison to blocks such as the OR, AND, and pass.

Our plan is, by compromising area for speed when it comes to the adder, with the adder having the longest delay, we will be able to actually decrease the area of the other blocks to match the delay of the adder. Because it is the longest delay that will control the clock speed, having blocks that are faster will just create a wait for the next clock edge. As long as all delays meet criteria such as set up and hold times, we can decrease the area everywhere else that we used to improve the speed of the adder. Because the metric we are using is determined by (Active Power)*Delay^2*Area, delay has the highest weight, and is therefore most important to improve on.

For our implementation of the adder, we attempted to recreate our own version of a kogge-stone adder. In an adder, the carry is always the determining factor for the worst case delay. The kogge-stone adder is an implementation of a carry look-ahead adder,

sacrificing number of transistors, and hence area, for speed. This adder should generate the carry bit in O(log n) time compared to O(n) for a ripple carry, where n is the number of bits.

The way we implemented the two's complement is using a two stage process, all bits being inverted first, and then passed through the adder with a 1. As part of the higher level decision, we will, for our final design, create a separate two's complement component and adder component because we do not want to waste the area having two separate adders in our ALU. Instead, we create an inverter block and feed the output to the adder we already have. To do this, we need to make sure that when you choose to implement the two's compliment, you always only have one path possible, without signals fighting for a position at a node. For this, we set controls for the input for the adder to choose between the inputs for the adder itself, or the pass from the inverter block and a static one.

Moving forward, we will implement an XOR gate for our arbitrary gate. Furthermore, once we have the best possible delay for the adder, will be to adjust the sizing for all the other transistors, keeping the ratios the same, but to actually increase the delay and decrease total area towards a better metric. This will take some testing using ocean scripts, with size being a metric of the transistor rather than a variable we can sweep in cadence alone. We will also need to compile all the components, including the registers and begin testing with controls for the ALU itself and making sure the entire system works the way we expect. In terms of delay we do need to make sure that the (W/L) ratios have the appropriate values for the worst case delay we calculate. This is very important in keeping the currents equal. For example, for our OR gate our (W/L) ratios for the pmos transistors consisting of the pull up network of the NOR portion of the gate needs to be 4 times the width of the nmos transistors in the inverter.

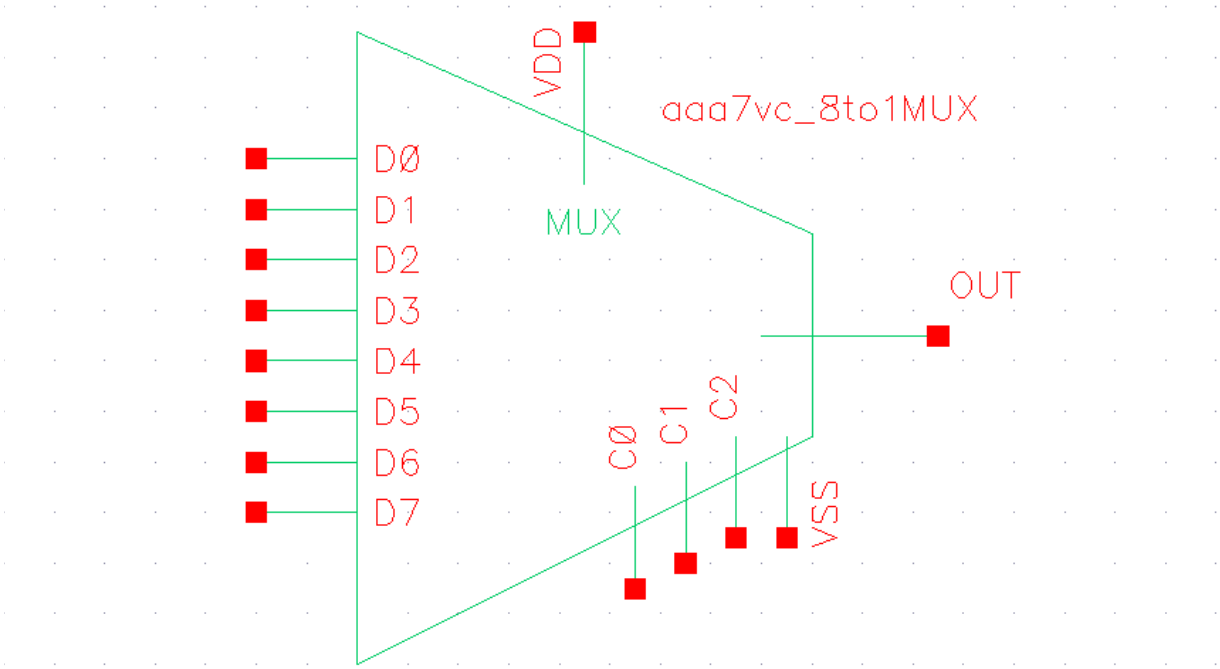
To actually test our final product we will have a circuit that incorporates the registers and ALU where we know that the circuit works perfectly and we will have an array of inputs

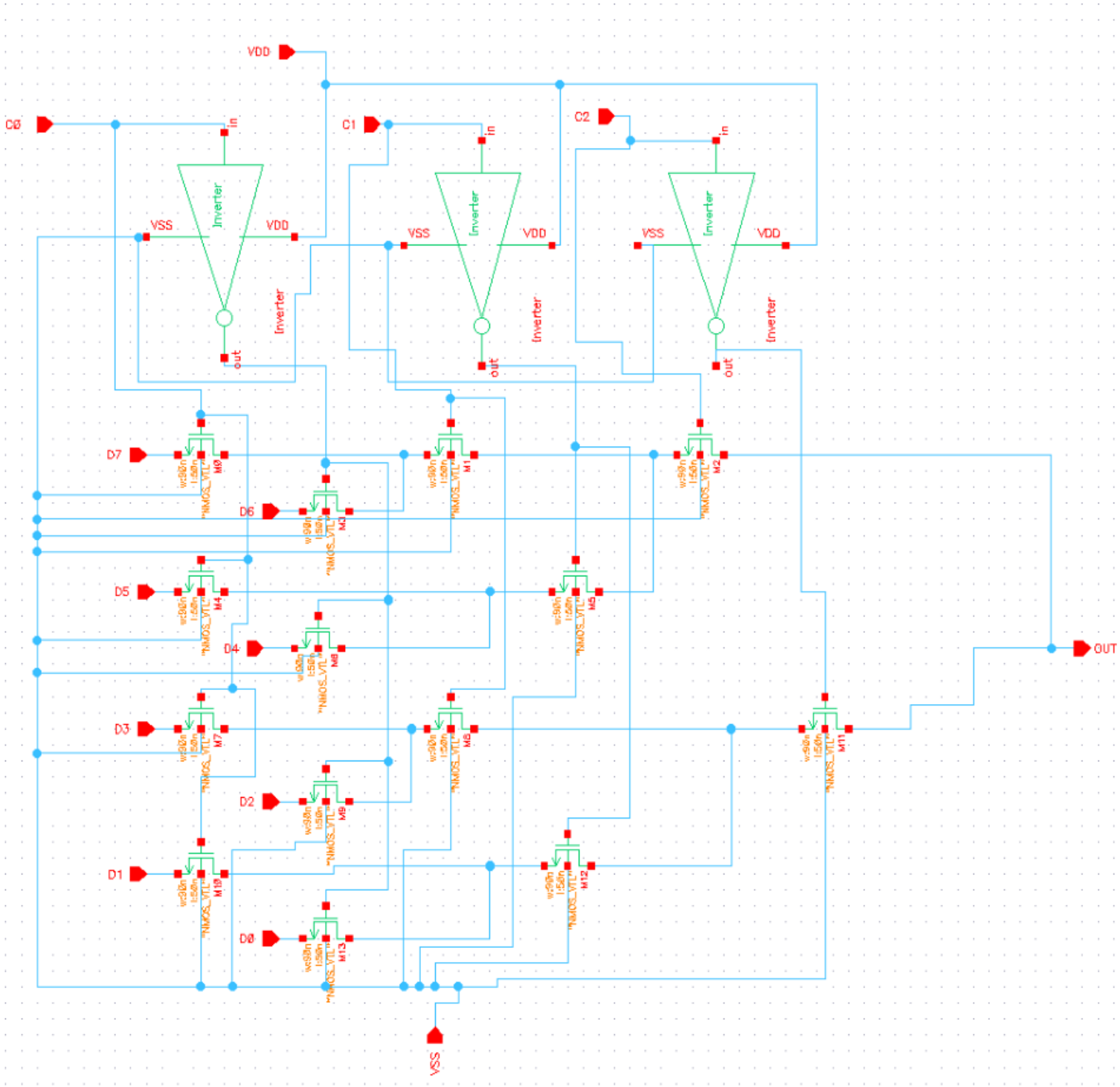
which we will feed to our design and to the “perfect” design. We will then run the outputs from both circuits to an XOR gate and if our design works as specified the XOR should return a 0 (since the values will match).

The top level design for our project is as follows:

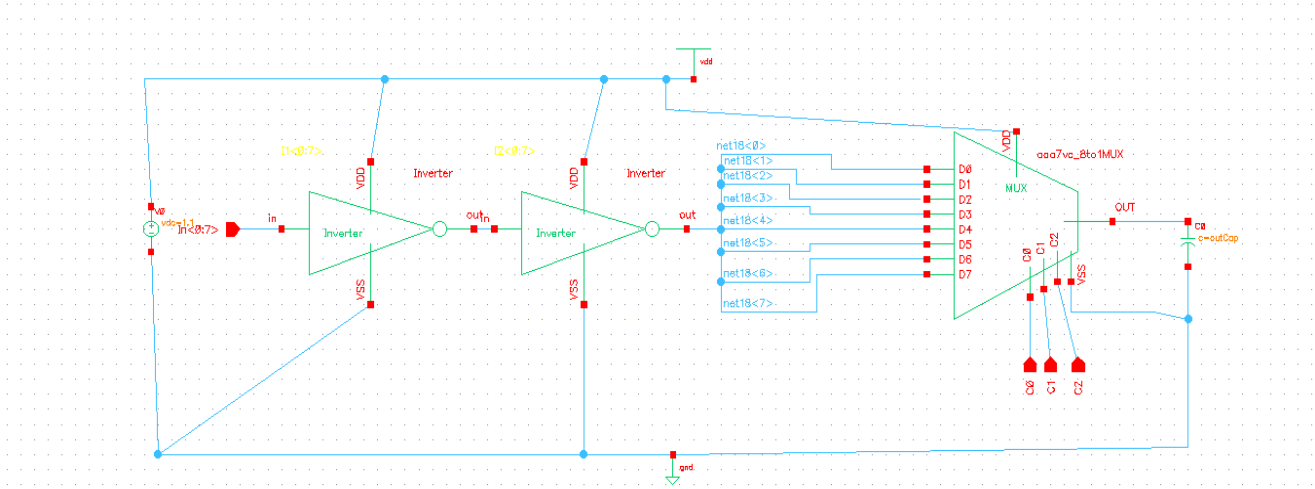
ALU COMPONENTS:

8-1 Mux:



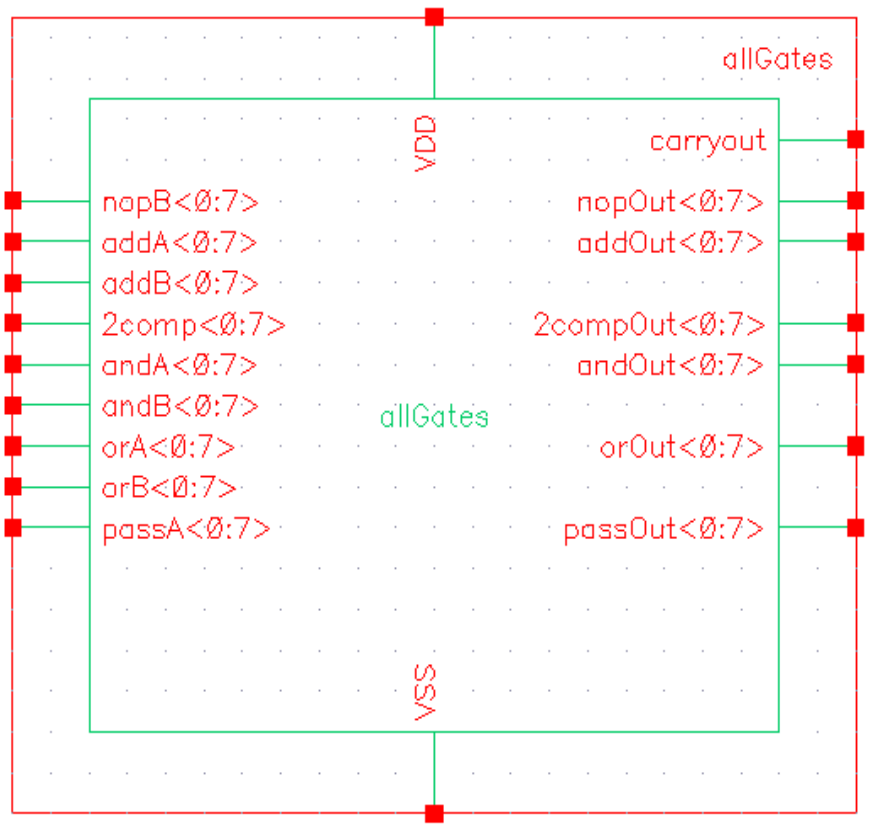

VDD
C0
C1
C2
in
Inverter
VSS
VDD
out
D7
D6
D5
D4
D3
D2
D1
D0
OUT
VSS

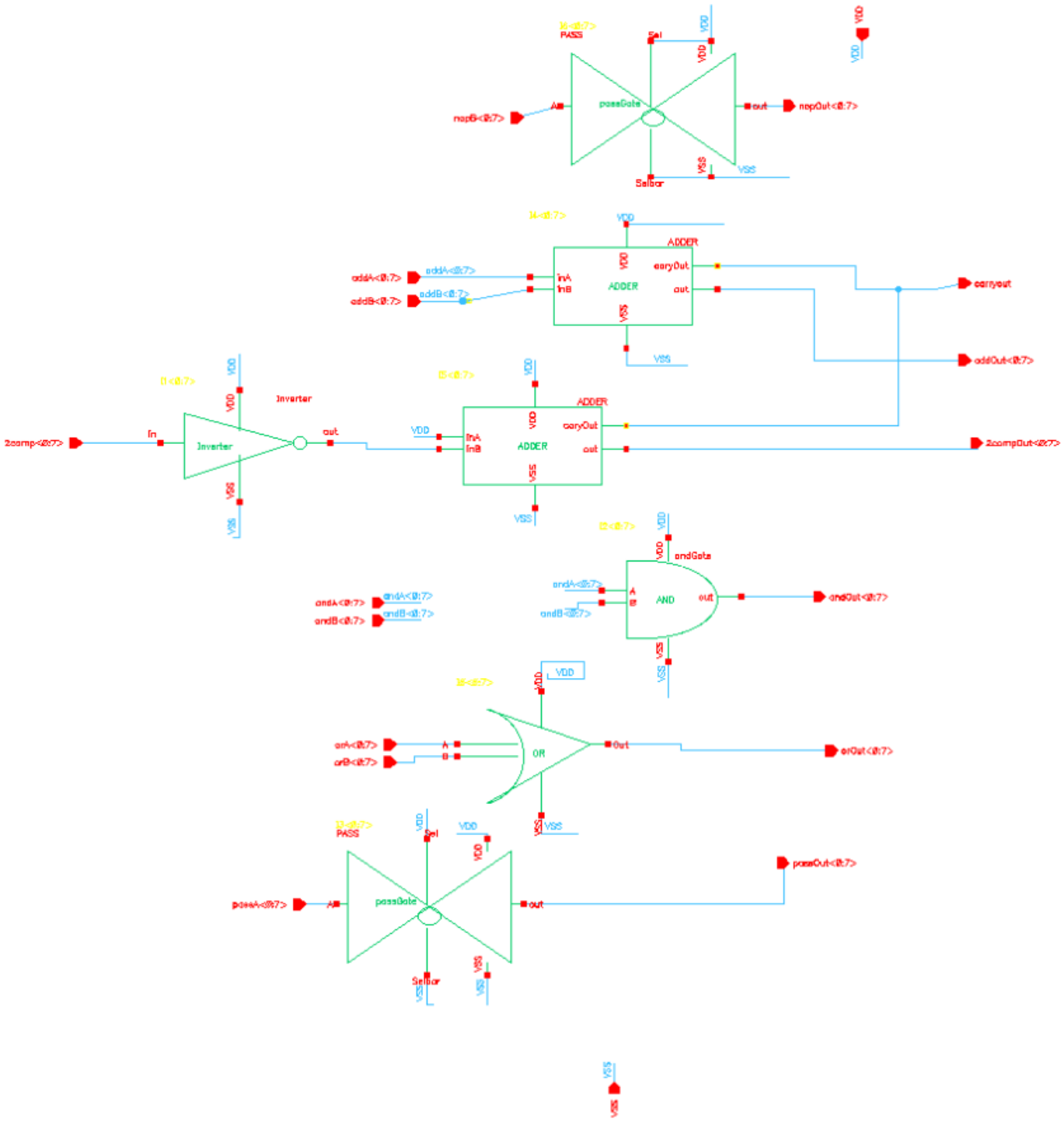
Mux Test Bench:



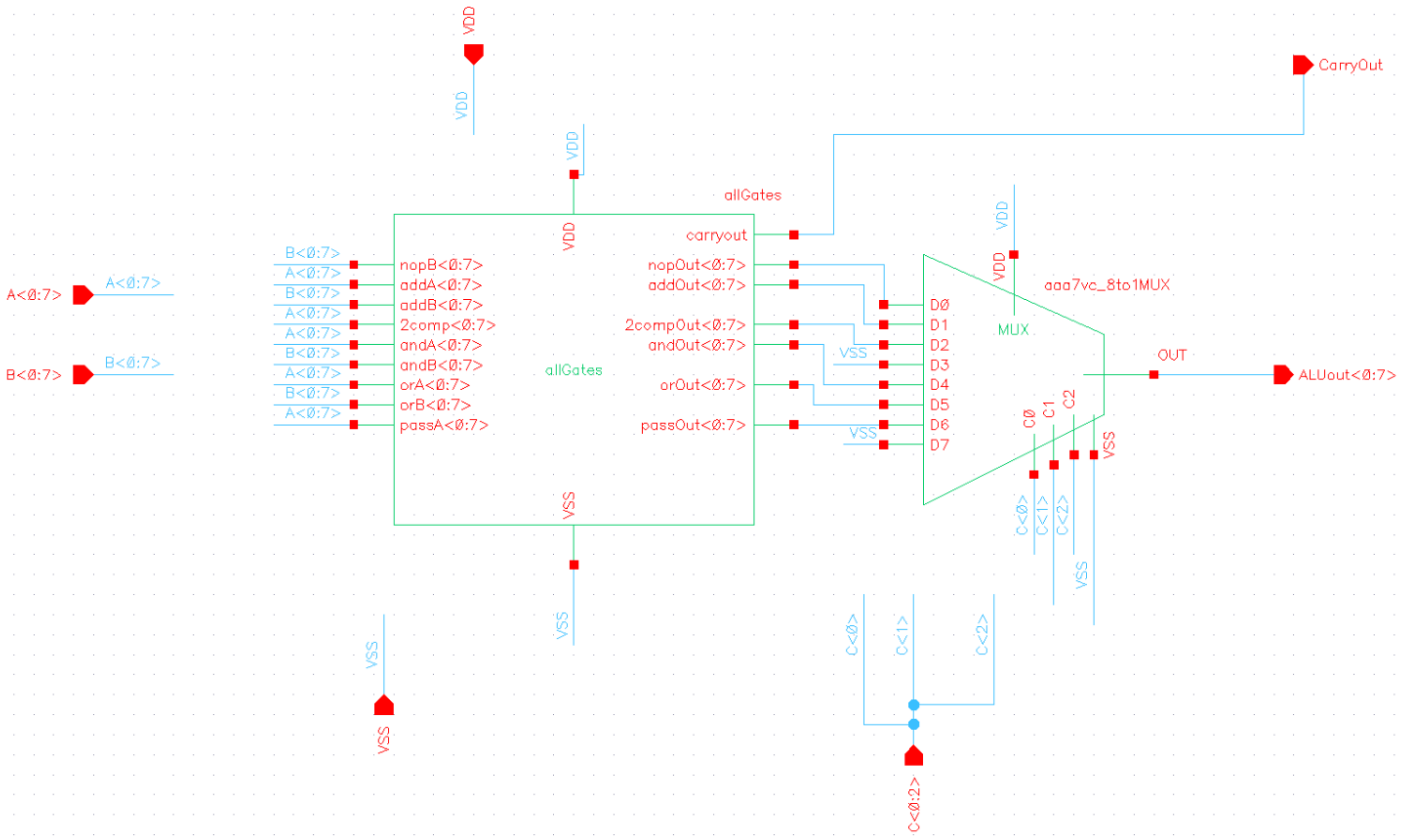
Other than the MUX, as part of the ALU we built a component called “Gates” that is just an abstraction of all the gates just to remove clutter from the actual ALU. It takes the inputs and performs the designated functions and outputs to the MUX, which chooses what to output.

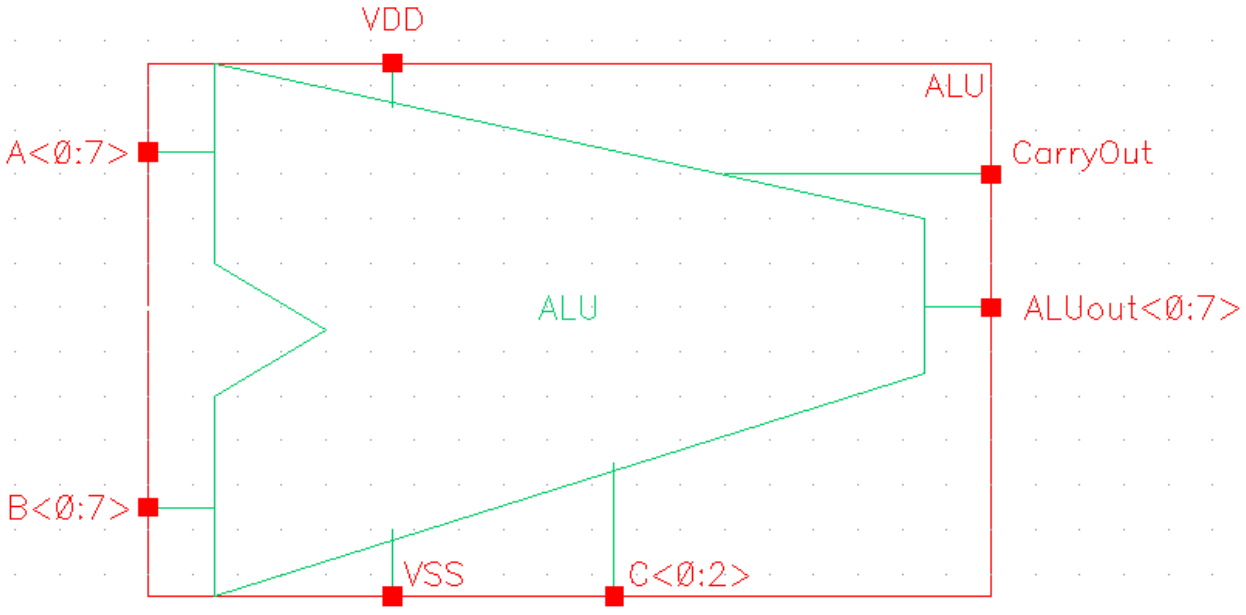
Gates:




PASS
Sel
VDD
passGate
A
out
nopB<0:7>
nopOut<0:7>
Selbar
VSS
ADDER
InA
InB
carryOut
addA<0:7>
addB<0:7>
carryout
addOut<0:7>
Inverter
In
2comp<0:7>
2compOut<0:7>
andGate
AND
andA<0:7>
andB<0:7>
andOut<0:7>
OR
orA<0:7>
orB<0:7>
orOut<0:7>
passA<0:7>
passOut<0:7>

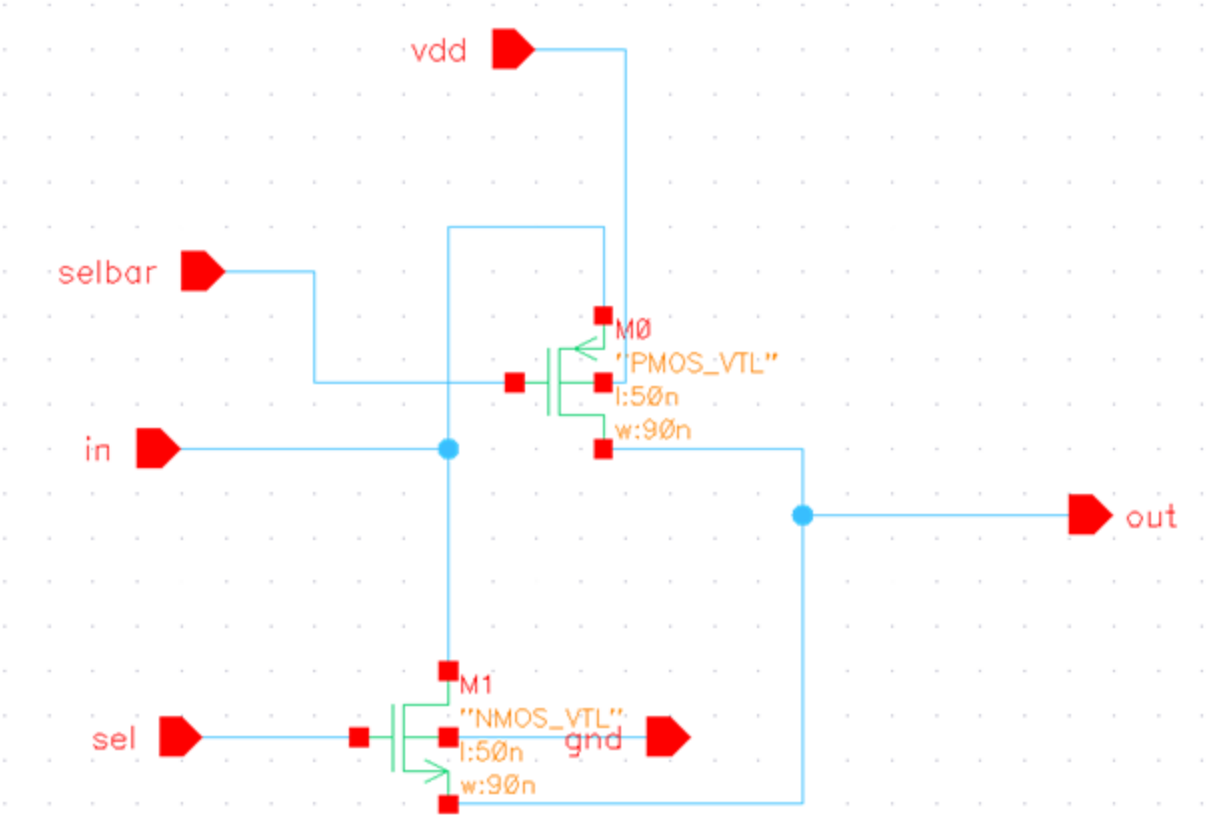
Finally these components are put together to form **the ALU**:

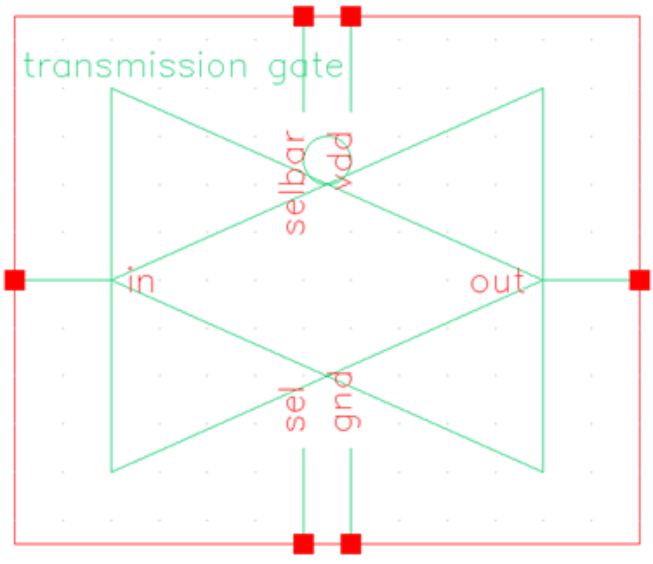




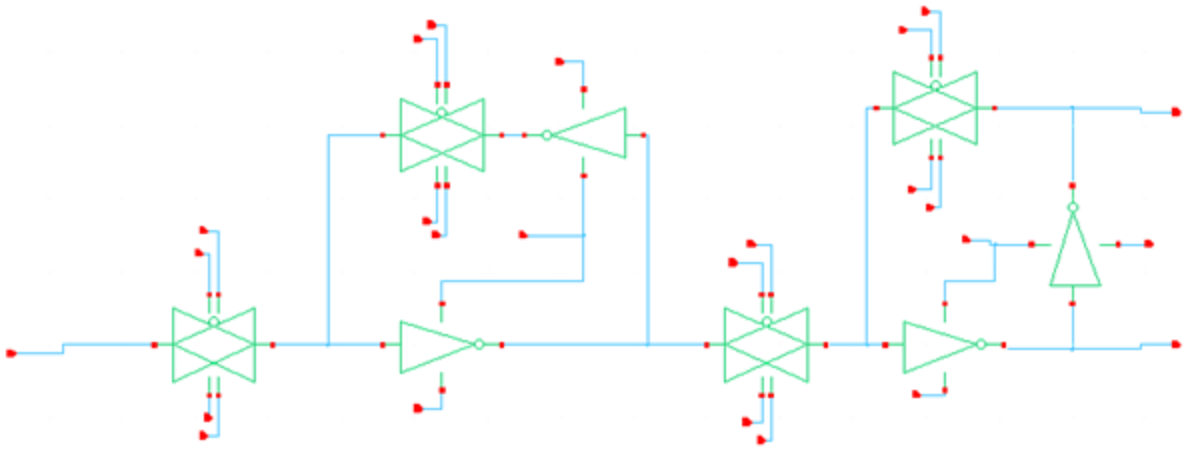
To connect our ALU we will need clocked registers. These registers can be realized using D Flip-Flops:

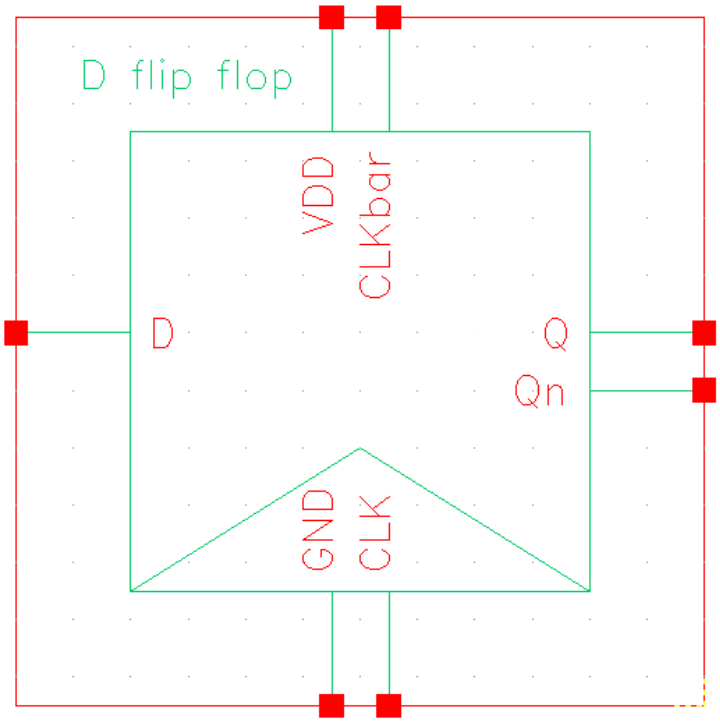
Transmission Gate circuit schematic and symbol:



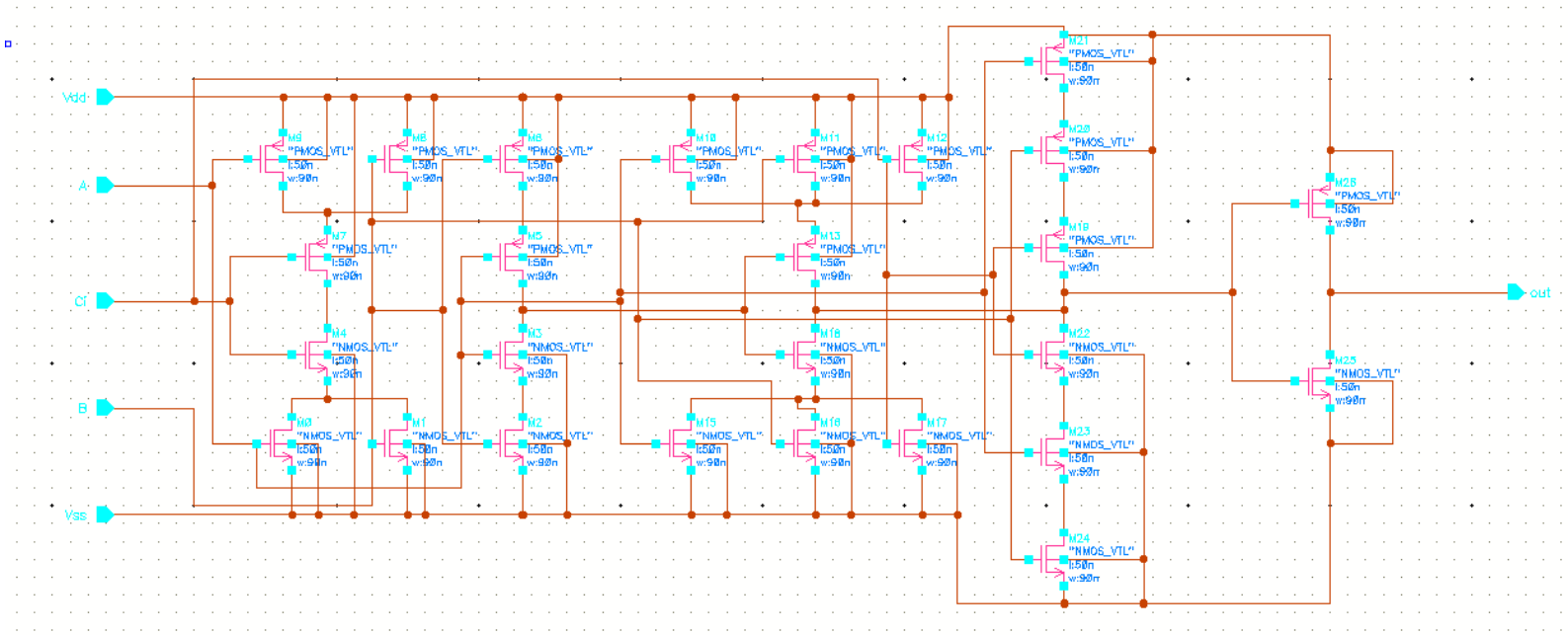


D flip flop circuit schematic and symbol:





As stated earlier we constructed our 2's complement using an inverter and an adder. We show both circuits below.



The figure above represents a two full bit mirror adder. the 8bit version consist of putting together 8 2bit adders as described in the figure below using symbols.is possible to create a logical circuit using multiple full adders to add N-bit numbers. Therefore, in our eight bits adder, in order to have one output that will sum all the 8 inputs, we need we have 7 two bits Full adder of operators and one bit of input to carry. The next figure below illustrates the general concept of a full 8bit adder.

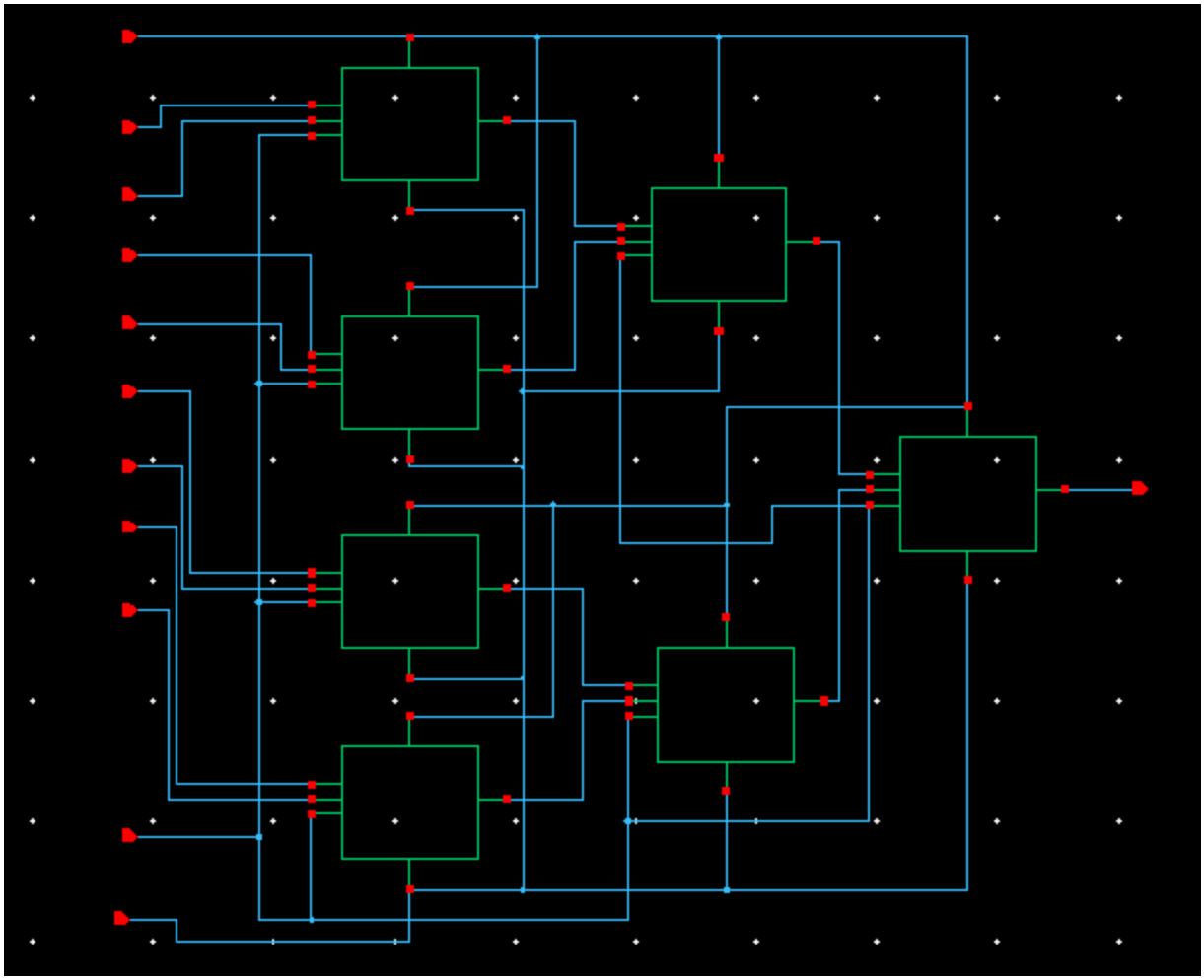
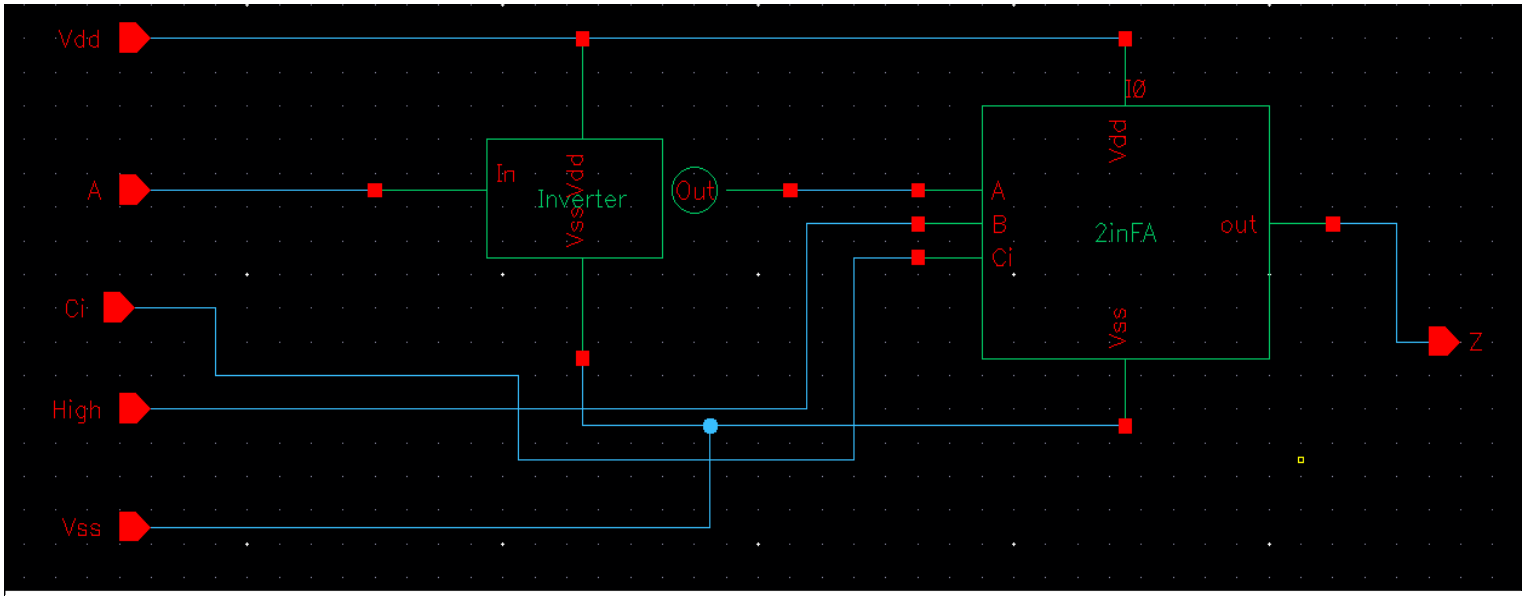


Figure of a full 8bit adder



One way to test the 8 bits two's complement is to use the instance A<0:7>. Further simulation will be made to test the functionality.